Deployment of Flower-CDN

Manal El Dick, Esther Pacitti

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Deployment of Flower-CDN

Manal El Dick — Esther Pacitti

N° 7129

Décembre 2009

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--7129--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
RENNES - BRETAGNE ATLANTIQUE

Deployment of Flower-CDN*

Manal El Dick †, Esther Pacitti‡

Thème COM — Systèmes communicants
Équipe-Projet Atlas

Rapport de recherche n° 7129 — Décembre 2009 — 16 pages

Abstract: Flower-CDN is a *Peer-to-Peer Content Distribution Network* (P2P CDN) that enables any under-provisioned website to distribute its content by relying on the community interested in its content. Therefore the deployment of Flower-CDN is supported by the clients that are willing to contribute in behalf of the website of their interest. This report provides the first guidelines to make Flower-CDN available for public use. We propose to implement Flower-CDN functionality as an extension for the user's web browser. As such, the user enjoys a transparent, flexible and highly configurable experience with Flower-CDN. Furthermore, we design an implementation architecture that covers security and privacy issues in a simple and practical manner.

Key-words: P2P, CDN, deployment, browser extension, implementation

* Work partially funded by the DataRing project of the french ANR.
† INRIA et LINA, Université de Nantes
‡ INRIA et LIRMM, Université de Montpellier 2

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu, 35042 Rennes Cedex
Téléphone : +33 2 99 84 71 00 — Télécopie : +33 2 99 84 71 71

Déploiment de Flower-CDN

Résumé : Flower-CDN est un réseau de distribution de contenu pair-à-pair qui permet à tout website de distribuer son contenu par l'intermédiare de la communauté intéressée à son contenu. Le déploiment de Flower-CDN est assuré par les clients qui sont disposés à participer afin de profiter d'un meilleur accès au contenu de leur intérêt. Pour une participation flexible et transparente, nous avons choisi d'implémenter la fonctionalité de Flower-CDN via une extension qui pourrait être intégrée au navigateur web du client. Dans ce rapport de recherche, nous décrivons l'extension et proposons l'architecture d'implémentation.

Mots-clés : P2P, CDN, déploiement, extension navigateur, implémentation

1 Introduction

Flower-CDN [1, 2] is a *Peer-to-Peer Content Distribution Network* (P2P CDN) that does not require dedicated or powerful servers. Flower-CDN distributes the popular content of any under-provisioned website by stricly relying on the community of users interested in its content. To achieve this, it takes into account the interests and localities of users, and accordinly organizes peers and serves queries. Flower-CDN adopts a novel and hybrid architecture that combines the strengths of the two types of P2P networks, i.e., structured and unstructured. It relies on a P2P directory service called D-ring, that is built and managed according to the interests and localities of the peers providng its services. D-ring helps new participants to quickly find peers in the same locality that are interested in the same website. Peers with respect to the same locality and website form together a cluster overlay called *petal*, to enable an efficient collaboration. Within a petal, peers use gossip protocols [3] to exchange information about their contacts and content, allowing Flower-CDN to maintain accurate information despite dynamic changes. We use this two-layered infrastructure consisting of D-ring and the petals for a locality-aware query routing and serving. D-ring ensures a reliable access for new clients, whereas petals allow them to subsequently perform locality-aware searches and provide them close-by content. Thus, most of the query routing takes place within a locality-based cluster leading to short response times and local data transfer.

Flower-CDN is deployed over clients that are interested in some particular website and that are willing to participate in order to enjoy a better access for the content of their interest. A website ws is supported by Flower-CDN as long as there are a sufficient number of clients on behalf of ws. More precisely, the more popular a website ws is, the more participants are attracted to Flower-CDN to populate the petals of ws and to occupy its directory peer positions. As for an unpopular website, its petals tend to be empty and its directory peer positions vacant.

A user accesses the Web through its web browser which handles her HTTP requests and accordingly allows her to search and view web content. In order to use and contribute to Flower-CDN transparently, a user should incorporate Flower-CDN functionality into her browser and let it run over HTTP.

We propose a Flower-CDN browser extension that enables P2P content distribution in a transparent and flexible manner. Additionally, it provides configuration management through which users can dynamically update their interests and enforce privacy preferences by specifying which content they will share. Finally, we adopt a simple security model that guarantees content integrity even in the face of untrusted peers.

Roadmap. In the following, we first introduce how Flower-CDN can be integrated into the user's web browser in Section 2. Then, we deepen our study by describing the implementation architecture of Flower-CDN in Section 3. Finally, we conclude in Section 4.

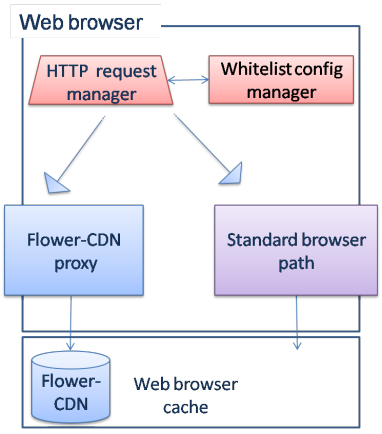

Figure 1: Flower-CDN extension within the web browser.

2 Flower-CDN Browser Extension

Flower-CDN functionality can be implemented as a browser extension. Figure 1 illustrates the changes affecting a browser that installs Flower-CDN extension. Flower-CDN operates via three main components: a *whitelist config manager*, an *HTTP request manager* and a *Flower-CDN proxy*.

As shown in Figure 1, the content that the user shares in Flower-CDN is stored in a delimited section of the browser cache (i.e., the disk storage allocated for the web browser). This ensures the privacy of the user, because it allows to isolate the web content that the user wants to share from the private content. The amount of disk space allocated to Flower-CDN section grows dynamically as more content is cached, bounded by the available disk space of the browser cache. The cache replacement and expiration policies adopted by the browser cache are used to manage Flower-CDN content (recall that the content mainly consists of web pages and their embedded objects). Further, the view and directory informations of a peer are also stored in this Flower-CDN section and managed according to their own expiration policies (i.e., the view via gossip exchanges and the directory information via push and keepalive).

The *whitelist config manager* mantains a list configured by the user and called *Flower-CDN whitelist* that specifies a set of domains referring to websites on behalf of which the user participates to Flower-CDN. The web browsing process begins when the user inputs a URL into the browser and initiates an HTTP request. This request is first handled by the *HTTP request manager*. It checks the URL against the *Flower-CDN whitelist* and forwards the request to the local *Flower-CDN proxy* if the URL matches the whitelist. Otherwise, the HTTP request follows the browser's standard processing path. Upon receiving the request, the Flower-CDN proxy tries first to locally resolve it and then resorts to the Flower-CDN network. The user is connected to the Flower-CDN network as a content or directory peer, via its local proxy which communicates with other Flower-CDN proxies at remote users. Thus, a Flower-CDN proxy handles requests coming from remote users in addition to the local user's requests.

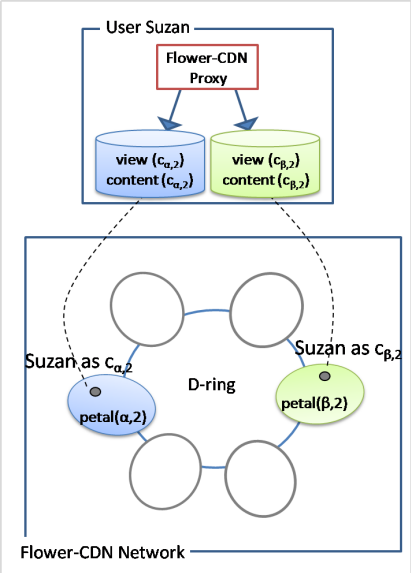

Figure 2: A user in Flower-CDN as two content peers related to two different websites.

Below, we first give more details on how a Flower-CDN extension is configured wrt. the user's interests and locality. Then, we give more explanation on how a user is connected to the Flower-CDN network (i.e., D-ring or petals).

2.1 Configuration

A user may have interest in several websites for which she wants to use Flower-CDN. In Flower-CDN, peers that are related to different websites are involved in different petals and thus have uncorrelated behaviors. Therefore, the user can participate in Flower-CDN as n different peers. She specifies, via the whitelist config manager, the names of the n websites of her interest and the cache section of her Flower-CDN proxy contains n subsections of dynamic sizes.

Figure 2 illustrates how a user Suzan is integrated in a Flower-CDN network. Suzan who is in locality 2 is interested in 2 websites α and β. Thus, she is represented in Flower-CDN network as 2 different content peers $c_{\alpha,2}$ and $c_{\beta,2}$. Technically speaking, the Flower-CDN proxy that operates within Susan's browser, manages two different cache subsections, one for each content peer. For instance, the first subsection contains the view and the content maintained by $c_{\alpha,2}$.

Upon the reception of an HTTP request, the Flower-CDN proxy detects the website ws targeted by the request based on its URL. If Suzan has a query for α, her Flower-CDN proxy accesses the Flower-CDN network as $c_{\alpha,2}$ and deals with the local cache subsection of $c_{\alpha,2}$.

Upon its installation by the user, a Flower-CDN browser extension is provided with the number k of localities involved in the system as well as the technique used to detect one's locality. For instance, if we use the landmark-based technique [4], the user will know the IP adresses of a set of well-known

landmarks spread across the network. Thus, she can measure its RTT to the landmarks and orders them by increasing latency. Given that each possible landmark ordering identifies a locality, the user detects her locality *loc* based on her ordering.

2.2 Connection with Flower-CDN network

Recall that a new client uses D-ring to enter Flower-CDN. Thus, a newly installed Flower-CDN browser extension has a list of IP addresses referring to random directory peers for bootstraping. When the Flower-CDN proxy wants to access Flower-CDN network for the first time, it uses a random bootstrap peer to route its first message over D-ring.

Upon receiving a query, the Flower-CDN proxy detects the target website ws and acts as the corresponding peer p. If this is the first query for ws, p needs to access D-ring. Thus, it computes the key reflecting the website targeted by the query and the locality of p and picks a random bootstrap peer which invokes the DHT routing procedure to forward the query to the target directory peer. If p has already submitted queries for ws, p acts as a directory or content peer of ws according to its acquired role, and uses its view to connect to peers from its petal hosted by remote users via their Flower-CDN local proxies.

A user may reconnect after a temporary disconnection or failure. In such a case, each one of her peers p does not necessarily have to take all the way via D-ring as if it is a new client. p can act as a content peer and try to renew contacts with other content peers of its petal using its previously built view which is stored within the user browser cache. More precisely, p searches for a contact from its view that is still available to gossip with in order for p to reintegrate its petal. However, if p's view contains no available contact, p cannot reintegrate its petal and thus has to rejoin Flower-CDN as a new client.

3 Flower-CDN Implementation

We now go inside the Flower-CDN proxy of each user and describe its implementation architecture. We first introduce the global architecture with its different layers. Then, we focus on Flower-CDN implementation details. To fully understand the details, one should first read Flower-CDN algorithms published in [1, 2]

3.1 Global Architecture

Flower-CDN relies on a DHT-structured overlay via its D-ring. Let us look at a standard DHT-based application. Then, we discuss Flower-CDN layering and identify the resulting changes.

3.1.1 DHT-based Applications

DHT systems provide an infrastructure for distributed storage and search. This is enbaled via two interfaces:

- $put(key, data)$ stores a key and its associated $data$ object in the DHT.
- $get(key)$ retrieves the $data$ object associated with key from the DHT.

Figure 3 shows the global architecture of a DHT-based application. On top of the Internet network there is the P2P overlay network that is structured as a DHT. The overlay layer ensures key-baesd routing and location by implementing the lookup method: *lookup*(*key*) returns the IP address of the DHT peer in charge of *key*. In order to guarantee the correctness of lookup, the overlay layer manages peer failures and churn by regularly updating routing tables [5]. On top of this layer, the distributed storage layer ensures key-based data distribution and search by implementing the put and get methods.

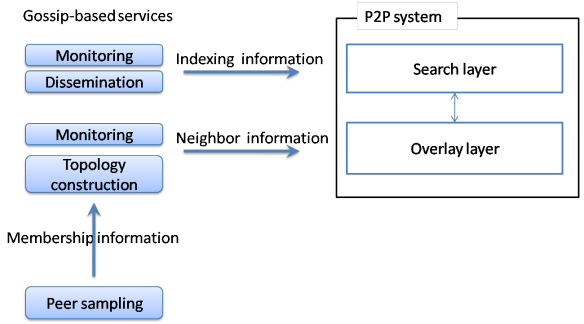

Figure 3: DHT-based application global architetcure.

3.1.2 Flower-CDN

Figure 4 illustrates a global architecture implementing Flower-CDN functionality.

The application layer now integrates two additional services, the HTTP request management and the whitelist config management.

The Flower-CDN proxy layer comprises two adjacent sublayers, the D-ring layer and the petal layer. The petal layer is represented by two main components that depict the behavior of a content peer, *content protocol* and *gossip protocol*. The D-ring layer consists of the key management service supporting the P2P directory service of D-ring. It comes directly over the DHT routing and location overlay. Flower-CDN does not use the distributed storage functionality of the DHT. Thus, the interfaces put and get are deactivated at each peer implementing Flower-CDN.

Flower-CDN only uses the routing and location overlay of the DHT to build the P2P directory service of D-ring. The DHT-based overlay provides the key-based lookup to route queries. Additionally, it is combined with stabilization protocols that keep routing tables up-to-date despite the arrivals and departures of peers [7, 6]. Basically, a peer periodically checks the liveness of its neighbors in the routing table. It also exchanges routing information with its neighbors to discover newly joining peers and accordingly update its routing table. Therefore, in Flower-CDN, the joins and leaves of directory peers are automatically detected by the DHT-based overlay.

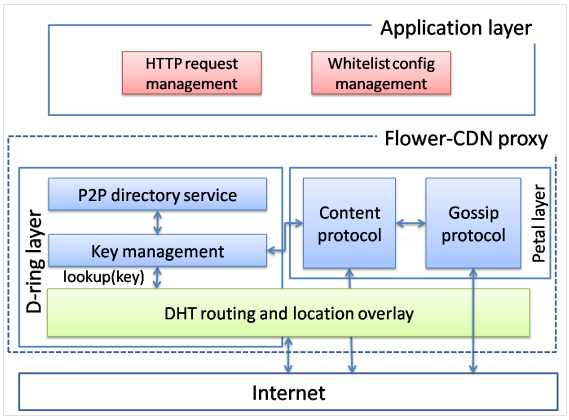

Figure 4: Flower-CDN global architecure

3.2 Implementation Architecture

The implementation architecture of Flower-CDN is illustrated in Figure 5. It shows seven components connected to links that refer to method invocations. Most of the links have two parts. The component that is connected to the part ending with a circle provides the method. The component that is connected to the part ending with an arc invokes the method. For instance, the component *content protocol* provides the method *processQuery*(q) that can be invoked by the component *interest manager*. In addition, some links have only one part and are only connected via the red circle to one component. This means that the component provides a method to be invoked by remote instances of the same component (e.g., via Java RMI). As an example, *content protocol* provides the method *processQuery*(q) for remote content protocols.

As introduced in Section 2.1, the Flower-CDN cache section is subdivided according to the interests of the user. In Figure 5, we omit this interest-based subdivision for simplicity. Further, we clearly separate the different types of data (i.e., view, content objects, directory-index, etc.) and include them in their appropriate component.

In this section, we first introduce the architecture components and then present in more detail their sequential interactions.

3.2.1 Components

Flower-CDN implementation architecture is organized under seven components: *locality manager*, *key manager*, *interest manager*, *directory protocol*, *content protocol*, *security manager* and *gossip protocol*.

Locality Manager. Its role consists in computing the own locality of the user. It offers one method:

- **getMyLocality**(): returns the locality of the current user.

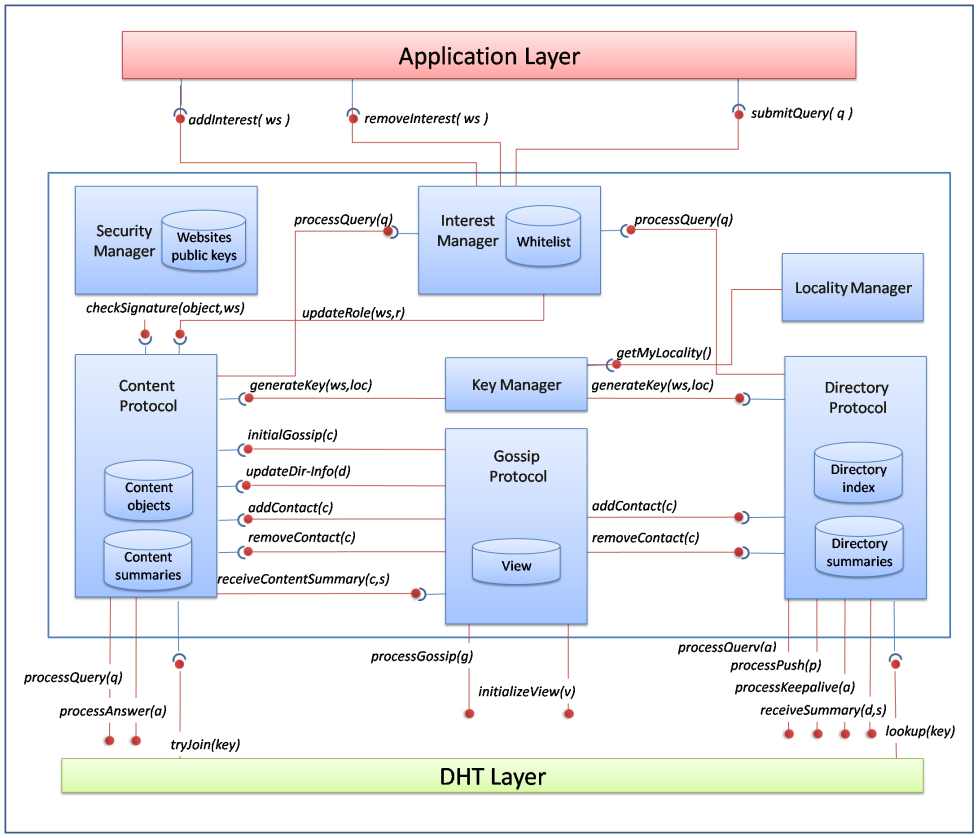

Figure 5: Flower-CDN implementation architecture.

Key Manager. It provides the key management service of D-ring. This component is used by the local content protocol and directory potocol when they need to access D-ring via the DHT layer. For this, it offers one method:

- **generateKey**(**ws**, **loc**): generates a key when provided with the url of the target website *ws* and the identfier of the target locality *loc*.

Interest Manager. It holds the whitelist specified by the user (i.e., the user interests in terms of websites for which she wants to contribute in Flower-CDN). It provides two methods for the user interface to update the whitelist according to the current user interests:

- **addInterest**(**ws**): adds the website *ws* to the whitelist.
- **removeInterest**(**ws**): removes the website *ws* from the whitelist.

In addition, the *interest manager* is responsible of receiving the user queries via:

- **submitQuery**(**q**): enables the user to submit a query *q* related to a website from the whitelist. It detects the target website *ws* and redirects the query to the component *directory protocol* if the corresponding peer is a directory peer or to the component *content protocol* if the corresponding

peer is a content peer or a new client (see Section 2.1). For this, it calls the method *processQuery*(q) provided by both components.

Recall that for each interest, the user is represented by a peer that has a specific role among directory peer, content peer and client. Upon adding an interest ws, the default role associated with ws is client. Eventually, the role changes with time according to the behavior of the peer. To update the role associated to each website of the whitelist, the folllowing method is provided:

- **updateRole**(**ws**, **r**): sets the role of the local peer related to ws ($r = directory, content, client$).

Directory Protocol. It depicts the behavior of each directory peer $d_{ws,loc}$ hosted by the user. For this, it maintains the data related to each $d_{ws,loc}$ which consists of the *directory-index*(ws, loc) and the *directory-summaries* received from directory peers $d_{ws,loc'}$ of other users (same website as the local directory peer but for different localities). For each method, the *directory protocol* detects the target website and therefore uses the corresponding data. For simplicity, we consider one directory peer $d_{ws,loc}$ hosted by the user (i.e, the user is only interested in website ws).

The *directory protocol* provides the following methods:

- **processQuery**(**q**): handles a query q. q refers to a query targeting $d_{ws,loc}$. Therefore, this method can be invoked by the *interest manager* for the user's own queries as a directory peer. In addition, it can be invoked by remote directory protocols that have used D-ring to route the query towards $d_{ws,loc}$.

- **processPush**(**p**): processes a push message p received by $d_{ws,loc}$ from one of its content peers $c_{ws,loc}$. This method is invoked by the remote content protocol corresponding to $c_{ws,loc}$.

- **processKeepalive**(**k**): processes a keepalive message k received by $d_{ws,loc}$ from one of its content peers $c_{ws,loc}$. This method is invoked by the remote content protocol corresponding to $c_{ws,loc}$.

- **receiveSummary**(**d**, **s**): receives and stores a directory-summary s from a directory peer d (i.e., a neighbor of $d_{ws,loc'}$ on the D-ring, $d_{ws,loc'}$). This method is invoked by a remote directory protocol corresponding to $d_{ws,loc'}$.

The *directory protocol* can invoke a method of the DHT layer:

- **lookup**(**key**): searches for the directory peer responsible of key. This is used to route a query over D-ring towards its target directory peer. Such a query is originated and sent by a new client (hosted by a remote user) that is using the curent directory peer $d_{ws,loc}$ as a bootstrap. The method *lookup*(key) connects the local directory protocol to the target directory protocol so that it can invoke *processQuery*(q) on the target component.

Content Protocol. It depicts the behavior of a content peer or a new client related to query processing. It manages the data related to each content peer $c_{ws,loc}$ hosted by the user, which consists of the content objects locally cached by $c_{ws,loc}$ and the *content-summaries* received from remote content peers $c'_{ws,loc}$. Upon running a method, the *content protocol* can detect the target website and thus can manipulate the corresponding data. For simplicity, we consider one content peer $c_{ws,loc}$ hosted by the user (i.e, the user is only interested in website ws).

The *content protocol* provides the method:

- **processQuery**(**q**): processes the query q. This method can be invoked by the *interest manager* for the user's own queries as a new client or a content peer. It can also be invoked by remote content and directory protocols related to the same petal as $c_{ws,loc}$.
- **processAnswer**(**a**): processes an answer a that is received for a previously submitted query of the user.

The *content protocol* can invoke a method of the DHT layer:

- **tryJoin**(**key**): tries to join D-ring as a directory peer using *key* related to its locality *loc* and website ws.

Security Manager. It ensures the integrity of the content that is transferred to the local user. In an open P2P environment, some peers may be malicious and corrupt the shared content. This problem can be easily solved if web-servers provide digitally signed certificates along with their content [8]. The security manager only needs the website public key to verify the digital signature of an object related to this website and received by the user from some content peer. This solution is indifferent to peer dynamicity and copes well with a loosely-trusted environment. To achieve this solution, it provides one function:

- **checkSignature**(**object**, **ws**): invoked by *content protocol* upon downloading new objects.

Gossip Protocol. It is responsible of the gossip behavior of a peer. It manages the view of every content or directory peer hosted by the local user. For simplicity, we consider one local peer hosted by the user.

The *gossip protocol* provides a method to implement a gossip exchange between two content peers:

- **initializeView**(**v**): sets v as an intial view of the local peer (new content peer). This is invoked by a remote gossip protocol when the local peer has recently joined its petal as a content peer.
- **processGossip**(**g**): processes a gossip message g. It is invoked by a remote gossip protocol to initiate a gossip exchange with a local content peer.

The *gossip protocol* also interacts with the local components *directory protocol* and *content protocol* to update the view of the associated directory or content protocol. The associated methods are as follows:

- **addContact**(**c**): adds a new contact c to the view of the local peer.
- **removeContact**(**c**): removes the contact c from the view of the local peer.
- **initialGossip**(**c**): initiates a new client c for gossip exchanges. It consists in sending to c a subset of the local view so that c can initialize its view and become a content peer. This is invoked by a directory or content protocol when they get in touch with a new client.
- **updateDir − info**(**d**): updates the *dir-info* which refers to the view entry of the local content peer referring to its current directory peer.
- **receiveContentSummary**(**c**, **s**): receives the content-summary s from another content peer c that shares the same petal as the local content peer.

3.2.2 Components at Work

We have introduced the Flower-CDN components individually. We now present how they work together by discussing two typical scenarios.

Scenario 1. In Figure 6, we consider the case where the component *content protocol* representing a content peer $c_{ws,loc}$ is invoked via $processQuery(q)$. Four users are involved in this scenario: the local user that hosts $c_{ws,loc}$, the remote user 1 that hosts the directory peer $d_{ws,loc}$ of the $petal(ws, loc)$, the remote user 2 that hosts another content peer $c'_{ws,loc}$ of the petal, and the remote user 3 that hosts the originator of the query q.

$c_{ws,loc}$ handles the query. In case the object o_{ws} is locally cached, $c_{ws,loc}$ answers the query and serves o_{ws} by calling $processAnswer(a)$ of the originator's content protocol. If the latter is a new client, $c_{ws,loc}$ invokes the method $initialGossip(c)$ of the local gossip protocol that also represents $c_{ws,loc}$. Eventually, the gossip protocol invokes $initializeView(subset)$ on the gossip protocol of the originator in order to initialize the originator's view with a subset of $view(c_{ws,loc})$.

Now, let us look at all the other cases where the object o_{ws} is not in the local cache. In case $c_{ws,loc}$ needs to query the content-summaries for the requested object o_{ws} and has not found any, it redirects the query to $d_{ws,loc}$ by invoking $processQuery(q)$ on the directory protocol of the remote user 1.

In case $c_{ws,loc}$ has found a content-summary related to $c'_{ws,loc}$ showing that the latter might have a copy of o_{ws}, $c_{ws,loc}$ redirects the query to $c'_{ws,loc}$ by invoking $processQuery(q)$ on the content protocol of the remote user 2.

However, if $c'_{ws,loc}$ is not alive (e.g., remote user 2 has disconnected), $c_{ws,loc}$ removes it from is view via its gossip protocol and redirects the query to the original web-server ws.

Scenario 2. In Figure 7, we consider the case where the component *content protocol* representing a content peer $c_{ws,loc}$ is invoked via $processAnswer(a)$. Two users are involved in this scenario: the local user that hosts $c_{ws,loc}$ and the remote user 1 that hosts the directory peer of the $petal(ws, loc)$, i.e., $d_{ws,loc}$.

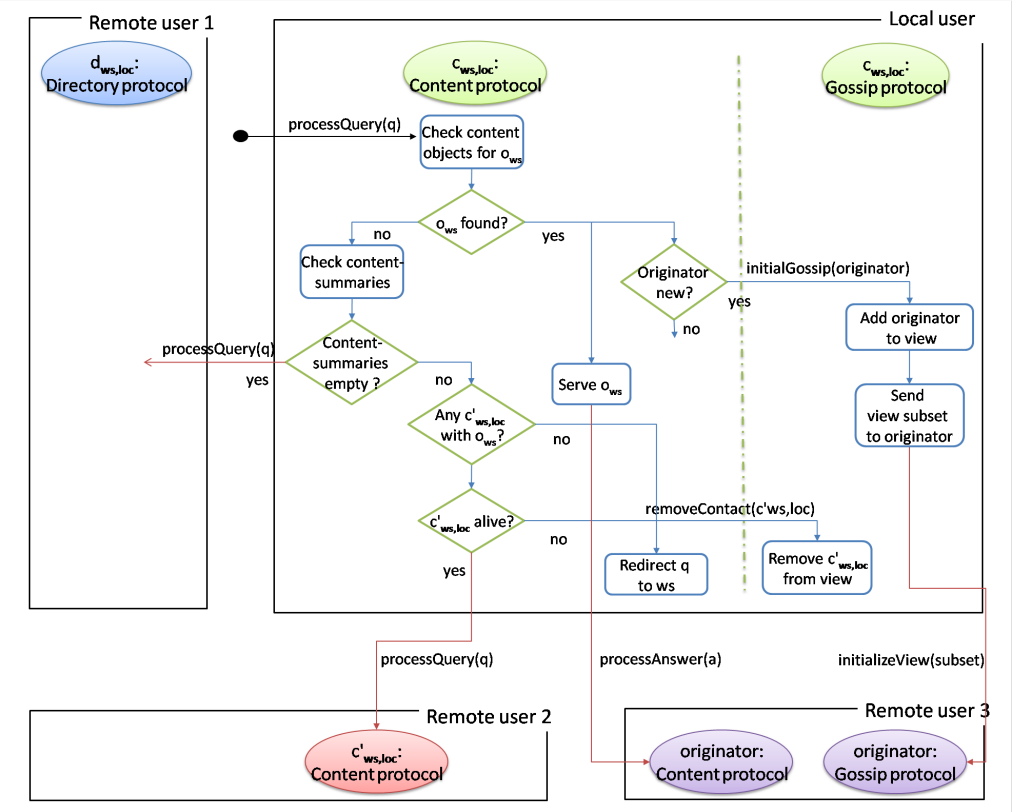

Figure 6: Scenario 1: $processQuery(q)$ at content protocol

Once a requested object o_{ws} is downloaded, $c_{ws,loc}$ checks its integrity by invoking $checkSignature(o_{ws}, ws)$ of the local interest manager. If validated, it caches the object. Then, it performs the push behavior: it checks if the number of changes in the local cache (i.e., content objects) has reached the threshold to notify the directory peer $d_{ws,loc}$. In such a case, $c_{ws,loc}$ sends a push message p to $d_{ws,loc}$ by invoking $processPush(p)$ on the directory protocol of the remote user 1. Upon contacting its $d_{ws,loc}$, $c_{ws,loc}$ updates its directory address information via $updateDir - info()$.

However, if $d_{ws,loc}$ is not alive (e.g., remote user 1 has disconnected), $c_{ws,loc}$ invokes the method $tryJoin(key)$ of the DHT layer in order to replace its directory peer. To obtain the appropriate key with respect to its locality loc and website ws, it invokes $generateKey(ws, loc)$ of the local key manager. Then, it is redirected over D-ring towards the target position. If $c_{ws,loc}$ succeeds in replacing its directory peer, it informs the interest manager about it via $updateRole(ws, directory)$. Otherwise, i.e., if $d_{ws,loc}$ has been already replaced by another peer, $c_{ws,loc}$ invokes $updateDir - info()$ to store the address information about the new $d_{ws,loc}$.

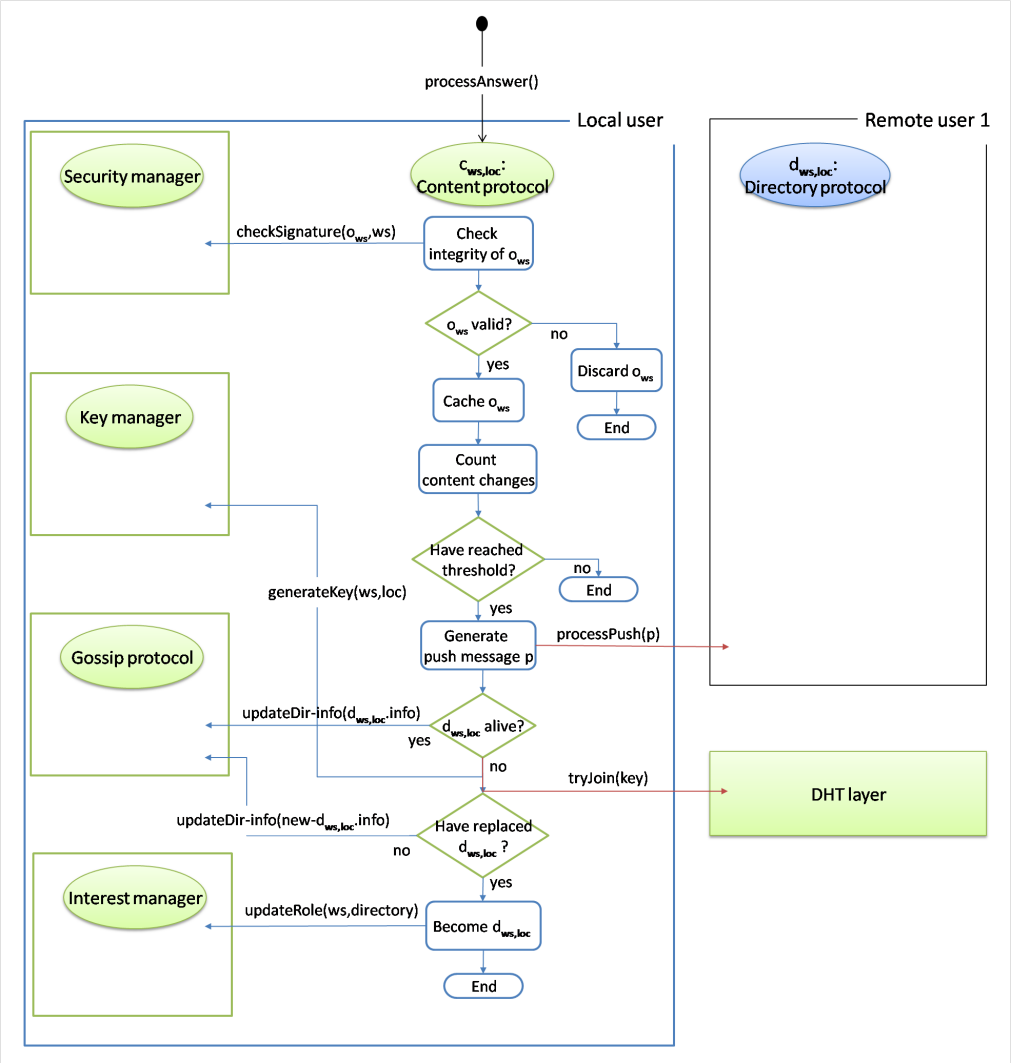

Figure 7: Scenario 2: *processAnswer*(q) at content protocol

4 Conclusion

In this report, we presented how Flower-CDN can be implemented and used in practice. The distinctive feature of Flower-CDN functionality is that it can be integrated into the user web browser, allowing the user to transparently contribute and exploit the benefits of Flower-CDN.

Furthermore, we showed that Flower-CDN functionality provides a highly flexible configuration for the user; it can easily manage the different interests of the user and their dynamic changes. We also sillustrated how the content authenticity can be ensured via a straigthforward solution. Finally, we designed the implementation architecture of Flower-CDN, which constitues an important step towards the implementation of Flower-CDN extension and its release for

public use. Then each interested user can simply download the extension and start using Flower-CDN.

References

[1] Manal El Dick, Esther Pacitti, and Bettina Kemme. Flower-CDN: a hybrid P2P overlay for efficient query processing in CDN. In *Proceedings of the 12th ACM International Conference on Extending Database Technology (EDBT)*, pages 427–438, 2009.

[2] Manal El Dick, Esther Pacitti, and Bettina Kemme. A highly robust P2P-CDN under large-scale and dynamic participation. In *Proceedings of the 1st International Conference on Advances in P2P Systems (AP2PS)*, pages 180–185, 2009.

[3] Patrick T. Eugster, Rachid Guerraoui, Anne-Marie Kermarrec, and Laurent Massoulieacute. Epidemic information dissemination in distributed systems. *IEEE Computer*, 37(5):60–67, 2004.

[4] Sylvia Ratnasamy, Mark Handley, Richard M. Karp, and Scott Shenker. Topologically-aware overlay construction and server selection. In *Proceedings of the 21st IEEE International Conference on Computer Communications (INFOCOM)*, pages 1190–1199, 2002.

[5] Sean C. Rhea, Dennis Geels, Timothy Roscoe, and John Kubiatowicz. Handling churn in a dht. In *Proceedings of the USENIX Annual Technical Conference, General Track*, pages 127–140, 2004.

[6] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale P2P systems. In *Proceedings of the 2nd ACM/IFIP International Conference on Middleware*, volume 2218 of *LNCS*, pages 329–350. Springer, 2001.

[7] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: a scalable P2P lookup service for Internet applications. In *Proceedings of the ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 149–160, 2001.

[8] Ian J. Taylor. *From P2P to Web services and Grids: peers in a client/server world*, chapter Security. Springer, 2004.

Contents

Centre de recherche INRIA Rennes – Bretagne Atlantique
IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Nancy – Grand Est : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399